

# Computer Science

**DEV.F**  
DESARROLLAMOS(PERSONAS);

dev

# Temas

- Introducción a POO.
- Objetos y clases en JS (prototipos).
- Sugar sintaxis para clases y objetos.
- Contexto de **this**.
- Call, apply y bind.



# Introducción a la POO

**DEV.F**  
DESARROLLAMOS(PERSONAS);

dev

# ¿Qué es la programación orientada a objetos?

Es un paradigma de programación, es decir, un estilo de programación que nos da unas guías sobre cómo programar los sistemas.

- Se basa en el concepto de clases y objetos.
- Busca dejar de centrarnos en la lógica pura de los programas, para empezar a pensar en objetos del mundo real.
- En vez de pensar en funciones, pensamos en las relaciones o interacciones de los diferentes componentes del sistema.
- Permite que el código sea reutilizable, organizado y fácil de mantener.

# Principios de POO

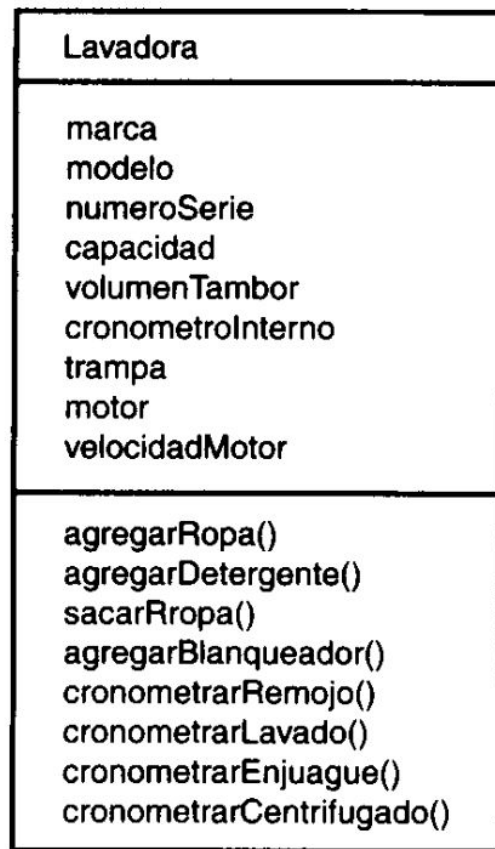
- **Abstracción:** Todo en el mundo real es un objeto.
- **Encapsulamiento:** Agrupar elementos dentro de un mismo nivel de abstracción.
- **Polimorfismo:** Los objetos pueden tener distintos comportamientos.
- **Herencia:** Las características y funciones de un objeto padre son heredadas por los hijos.
- **Modularidad:** Se separa en archivos llamados clases, las funcionalidades.
- **Principio de ocultamiento:** Los objetos sólo deben acceder a la información por medio de getters y setters.

# Conceptos

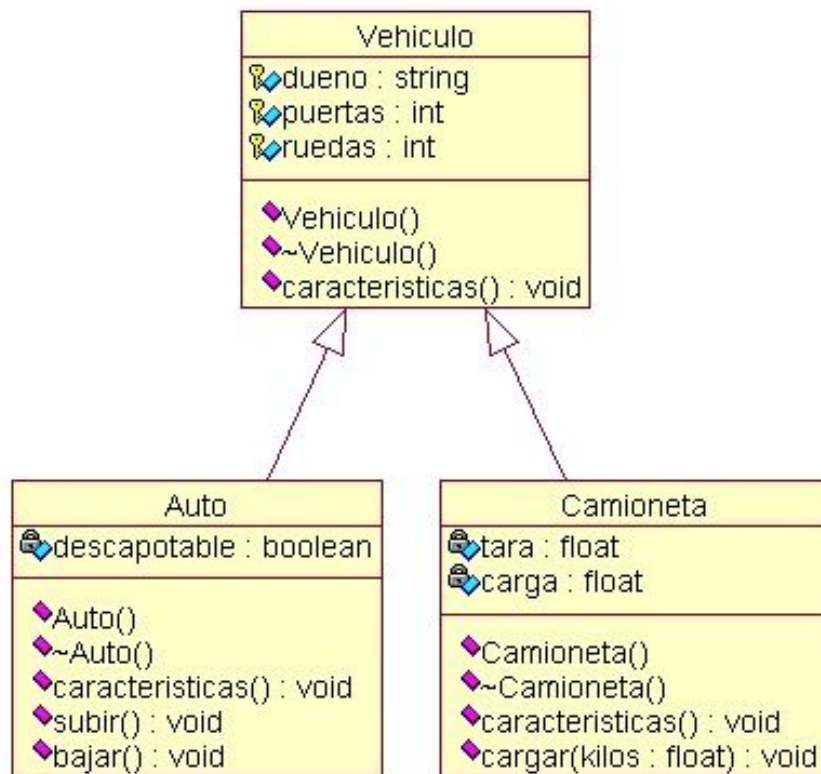
- **Clase:** **Modelo** o **plantilla** a partir del cual se crean objetos.
- **Objeto:** Es un **elemento concreto** del mundo real.
- **Atributo:** También se le conoce como **característica** o **propiedad**.
- **Método:** También se les conoce como **función** y son las acciones que el objeto de la clase puede realizar.



# Diagrama UML de una clase



# Diagrama UML jerarquía de clases





# Práctica 1

- Realizar el diagrama UML de una jerarquía de clases de autos.



# Creación de objetos en JS

**DEV.F**  
DESARROLLAMOS(PERSONAS);

dev

# Creación de objetos en JS

- **Objetos literales.**
- **Función constructora.**
- **Función constructora con new.**
- **Prototype.**
- **Sugar syntax.**

# JavaScript y la cadena de prototipos

**DEV.F**  
DESARROLLAMOS(PERSONAS);

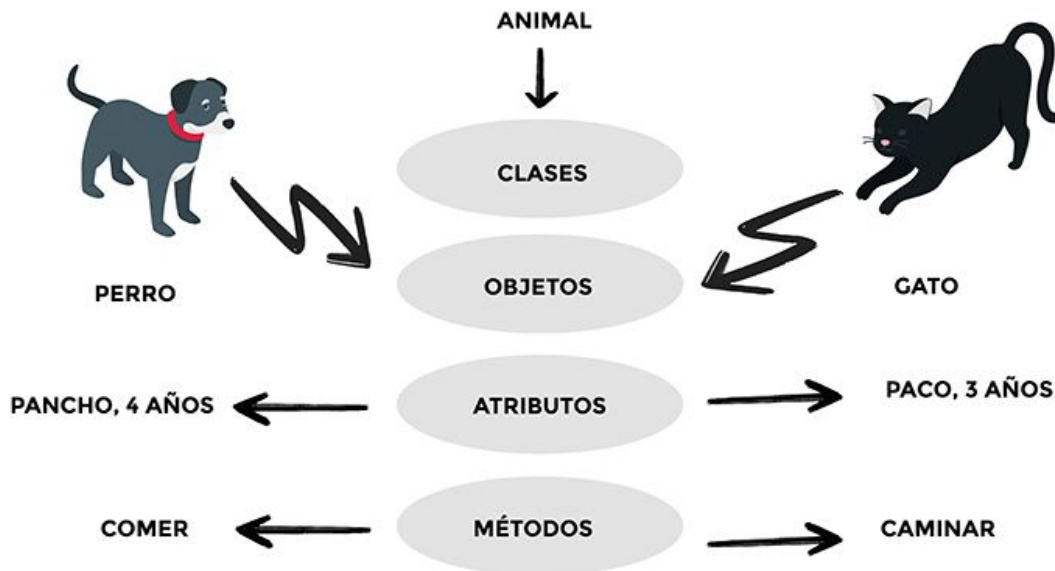
dev

# Principios de la cadena de prototipos

- **Prototipo:** Es homólogo a una clase.
- **Object:** El objeto de mayor jerarquía en JS.
- **Cadena de prototipos:** Simula el concepto de herencia mediante el objeto prototype.

# Lenguaje basado en prototipos

JavaScript es a menudo descrito como un lenguaje basado en prototipos. **Los objetos pueden tener un objeto base que utilizan como plantilla (prototipo)** para crear nuevos objetos y heredar métodos y/o propiedades.



# Cadena de prototipos

Un objeto prototipo del objeto puede tener a su vez otro objeto **prototipo**, el cual hereda métodos y propiedades, y **así sucesivamente**. Esto es conocido con frecuencia como la cadena de prototipos.



# Teoría de los Prototipos

- Los métodos y propiedades se agregan a la propiedad prototype, que reside en la función constructora del objeto.
- JavaScript establece un enlace entre la propiedad `__proto__` del objeto y su prototipo. **`Object.getPrototypeOf(obj) === obj.__proto__`**

## Ejemplo 1: primer prototipo.

- Las propiedades y métodos heredados están definidos en la propiedad prototype.
- Se puede acceder a la propiedad prototype del prototipo plantilla.



# Teoría de los Prototipos

- Cualquier prototipo de un objeto global está disponible en cuanto se crea una instancia de él.
- `Object.create()`: crea un nuevo objeto utilizando el objeto indicado como su prototipo.
- `.__proto__ = .constructor`
- Patrón de creación de objetos con funciones, propiedades dentro, métodos en el prototype.

# Métodos interesantes de prototipos

- `Object.getPrototypeOf(obj) === obj.__proto__`
- `obj.constructor.name`
- `.__proto__ = .constructor`
- `obj.hasOwnProperty(prop)`
- `obj.isPrototypeOf(obj)`
- `obj.is()`
- `obj.toString()`
- `obj.getOwnPropertyNames(obj)`
- `obj.hasOwn()`
- `obj.valueOf()`

## Práctica 2

Generar una jerarquía de clases con cada uno de los métodos mencionados.



# Sugar Syntaxis para clases y objetos

**DEV.F.**  
DESARROLLAMOS(PERSONAS);

dev

# Keywords para clases en JS.

- **Class.**
- **Extends.**
- **Constructor.**
- **This.**
- **Get y Set.**

## Práctica 3

Generar una jerarquía de clases con sugar syntax.



# Contexto de this

**DEV.F**  
DESARROLLAMOS(PERSONAS);

dev

## Práctica 4

Realizar practicar sobre el contexto de this en:

- Global.
- En una función.
- En una función con strict mode.
- En objetos.
- Desde fuera de objeto.
- A partir de un objeto nuevo creado por una función Constructora.





# Call, Apply y bind

**DEV.F**  
DESARROLLAMOS(PERSONAS);

dev

# Call, bind y apply

- **Call:** Permite establecer el contexto `this` con el que se llamará a una función y con argumentos provistos individualmente..
- **Apply:** Permite establecer el contexto `this` con el que se llamará a una función y con argumentos se informan como un arreglo.
- **Bind:** Crea una nueva función con un nuevo contexto `this`.

## Práctica 5

- Ejecutar un cambio de contexto de this con call, bind y apply.

